EXHIBIT B

To: Jim Kahle/Austin/IBM@IBMUS
cc: Charles Moore/Austin/IBM@IBMUS, Brian Konigsburg/Austin/IBM@IBMUS
From: Balaram Sinharoy/Poughkeepsie/IBM @ IBMUS
Subject: "IBM Confidential: Simplify GBH Update Logic ?

Hi Jim,

If we want there is a simplification we can make in the Global Branch History (GBH) update logic, but this has a performance impact for scientific code. This simplification works only when we do not have BTAC. In the absence of any exception to normal instruction fetching, we always fetch the next sequential sector.

An alternative (and much simpler) approach to update the GBH vector is to shift in "0" when we fetch the next sequential sector. On a BHT redirect (that is, there is a taken branch in the fetch group), we backup the GBH vector and then shift in "1" (logic diagram below). GBH update for handling other exception cases remains similar.

Functionally the major difference between POR and simpler approach is that thesimpler approach shifts "0" even if the fetch group does not have any branch in it. This lowers performance for scientific code. Another difference is that there is no two cycle delayed update of the GBH vector (as in POR). Two cycle delayed update has little performance impact, but due to it,the POR has more complex control and requires a moderate-sized state machine. Simpler approach has no state machine.

We did some performance runs on M2 model to compare the two approaches. Performance of the two

approaches are about the same for commercial workload. However, the alternate approach is substantially worse for scientific workload. For example, for APPLU (partial differential equation solver -- quite common in scientific code) we loose 5% CPI!.

APPLU (and other scientific code) typically has nested loops like the following, which simpler algorithm does not work well.

```
do i = 1 to 100000 (a large number)
  do j = 1 to 5 (a low number)
    number of matrix equations (but no branch)
  enddo
enddo
```

After two iterations of the outer loop, our POR GBH update will be able to predict the inner loop branch with near 100% accuracy. This is because, our algorithm learns that only when the 11-bit GBH vector is 01000010000 we will get another 1 (or taken). All other cases it will predict fallthrough. (We loose this advantage if the inner loop iterates more than 12 times).

Simpler approach dilutes the GBH vector with too many "0"s (for which there is no branch). If the loop body above has 3 fetch groups the simpler approach will shift 3 bits per inner loop iteration. Since five iterations forms a repetitive cycle, to remember all the five iterations, we will need 5*3 = 15 bit GBH vector. With 11-bit GBH vector (POR), we will mispredict 2 out of every 5 iterations (with 1-bit predictor, for each mispredict we set the BHT to wrong value causing another mispredict). In the worst case, when there is too many fetch groups in the loop body (but no branch) we can completely dilute the GBH vector with "0"s and end up with a result similar to just LBHT (which is quite poor for such loops). APPLU has such loop bodies (see example below, which is also one of the most frequently executed subroutine).

For commercial code, the simpler approach works well. Commercial code is very branch-oriented. In 11 successive fetch groups (1 bit in GBH per fetch group), there is almost always some branch. So if the prediction is path dependent, we almost never loose the path information. Moreover, the simpler algorithm might actually help in cases when we do not really need information from all the 11 previous branches. Some dilution in the GBH vector can actually reduce the same information from appearing in too many entries in the GBHT/GSEL tables. That means better use of the GBHT/GSEL tables and slightly better performance.

Thanks.

Regards,
Balaram

Performance Comparison (from Model2 ver 1.82)

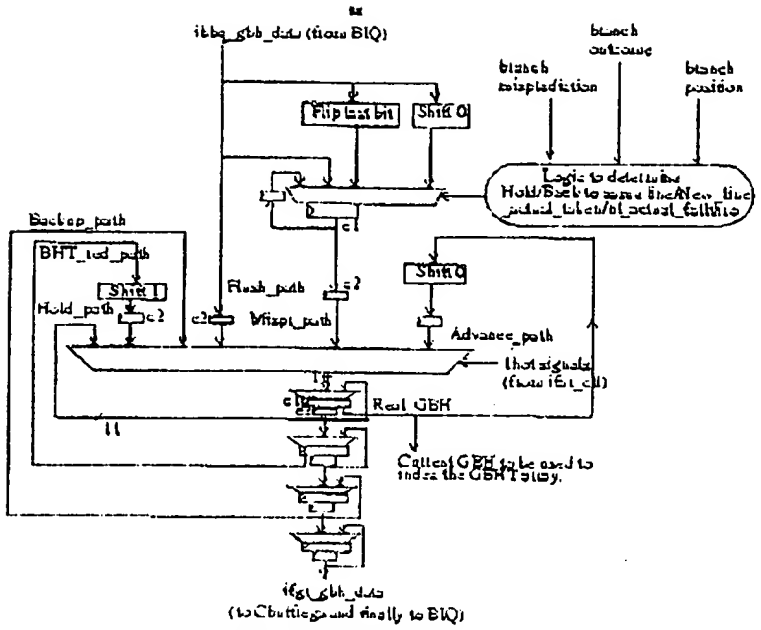
(POR numbers are very close to what we expected from static performance modeling)

	Branch_pred			CPI		
Workload	NewBrPred	PORBrPred	POR_over_New	NewCPI	PORCPI	POR_over_New
TPCC-DB2	95.17	95.00	-0.16	1.450	1.454	-0.22%
TPCC-ORA	95.18	94.96	-0.19	1.480	1.483	-0.22%

SPECFP Workload	NewBrPred	PORBrPred	Branch_pred POR_over_New	NewCPI	PORCPI	CPI POR_over_New
applu	94.79	99.84	5.05	0.905	0.861	5.07%
apsi	96.30	97.38	1.09	1.116	1.113	0.25%
fppp	91.80	95.19	3.40	0.869	0.863	0.74%
hydro2d	99.53	99.40	-0.12	1.439	1.441	-0.11%
mgrid	98.52	98.59	0.06	1.004	1.004	0.06%
su2cor	86.77	86.76	0.00	1.096	1.093	0.29%
swim	99.61	99.62	0.02	0.767	0.766	0.12%
tomcatv	98.56	98.42	-0.14	0.905	0.903	0.22%
turb3d	92.72	96.22	3.50	0.724	0.714	1.46%
wave5	98.91	99.46	0.56	0.990	0.987	0.26%
SpecFP_Avg	95.75	97.09	1.34	0.981	0.974	0.64%

SPECINT Workload	NewBrPred	PORBrPred	Branch_pred POR_over_New	NewCPI	PORCPI	CPI POR_over_New
compress	87.35	86.95	-0.39	1.136	1.141	-0.47%
gcc	93.21	93.20	-0.00	0.944	0.944	0.04%
go	83.53	82.91	-0.61	1.059	1.073	-1.38%
ijpeg	88.45	89.00	0.56	0.822	0.821	0.13%
li	95.20	95.65	0.46	0.737	0.727	1.51%
m88ksim	96.72	96.57	-0.15	0.880	0.883	-0.31%
perl	96.95	97.86	0.92	1.121	1.100	1.93%
vortex	99.41	99.45	0.04	0.688	0.687	0.15%
Specint_Avg	92.60	92.70	0.10	0.923	0.922	0.20%

Logic Diagram for Simpler Approach



Details of the Alternate GBH Update Control